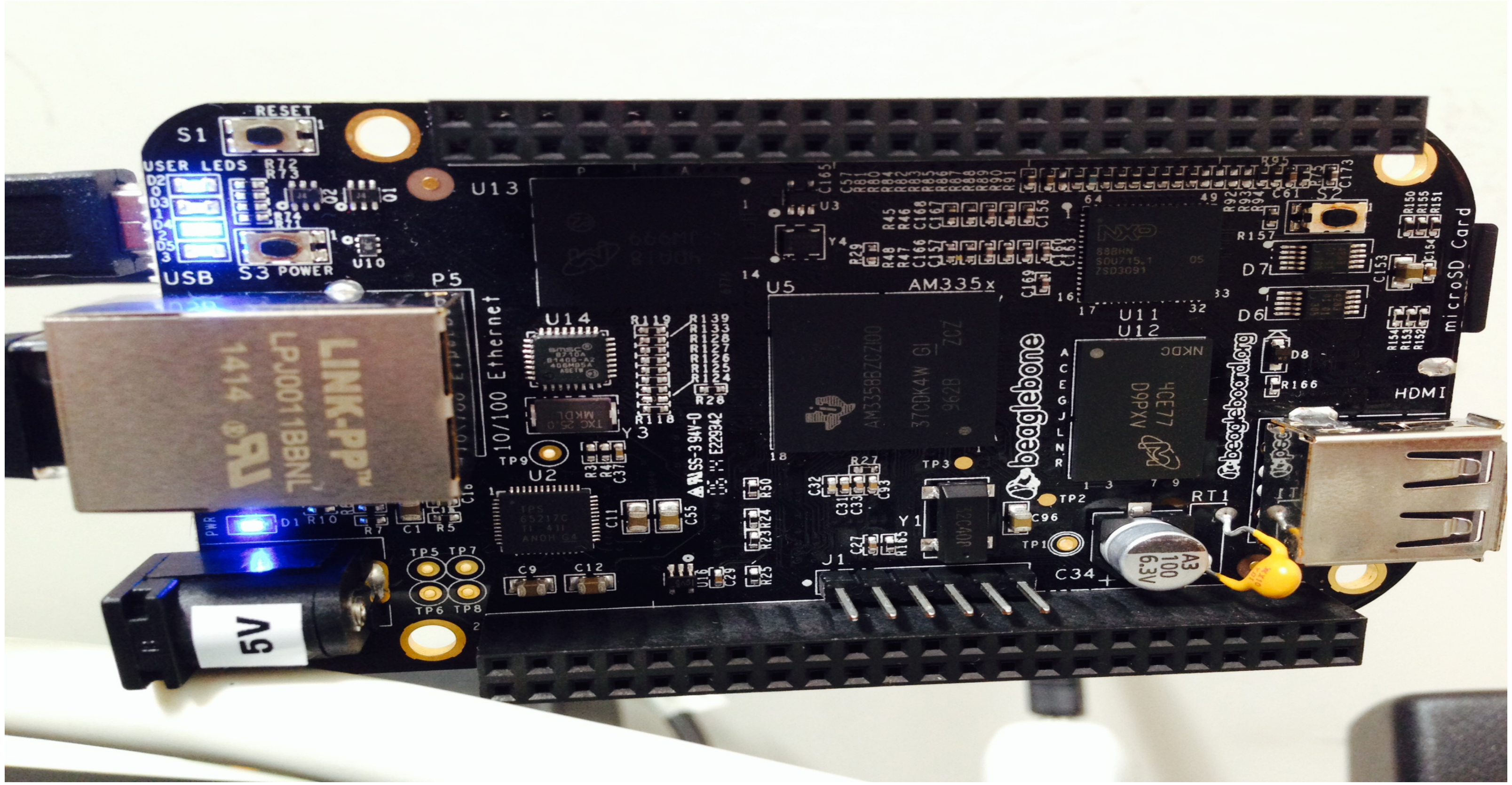
NGINX Powers 12 Billion Transactions per day at Capital One

Rohit Joshi

Mike Dunn

We built a prototype of an API Gateway using a BeagleBone Black, NGINX and Lua



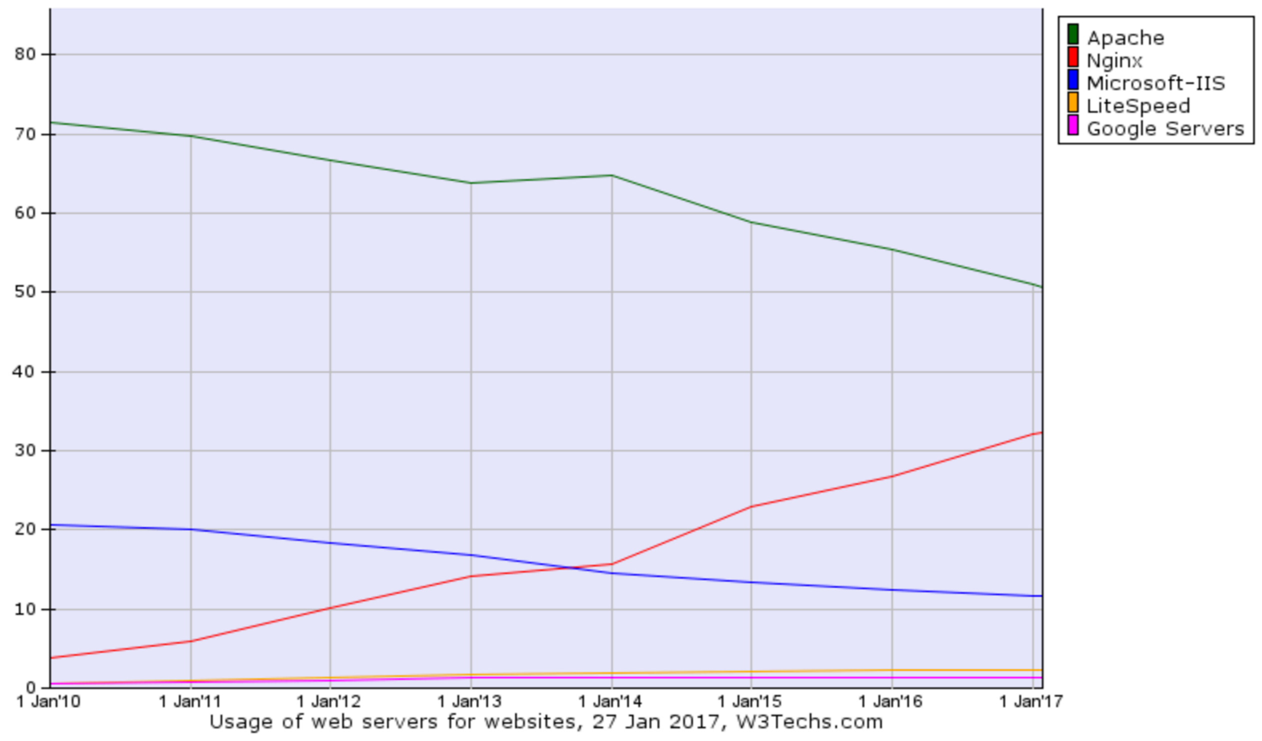
- Beagle Bone Black (5v power)
- ARM Processor
- The prototype was able to sustain:
  - 625 TPS for HTTP
  - 335 TPS for HTTPS
- Nginx PR: Support for mutual SSL authentication for upstream https proxy

What Technologies did we select for the build and why?

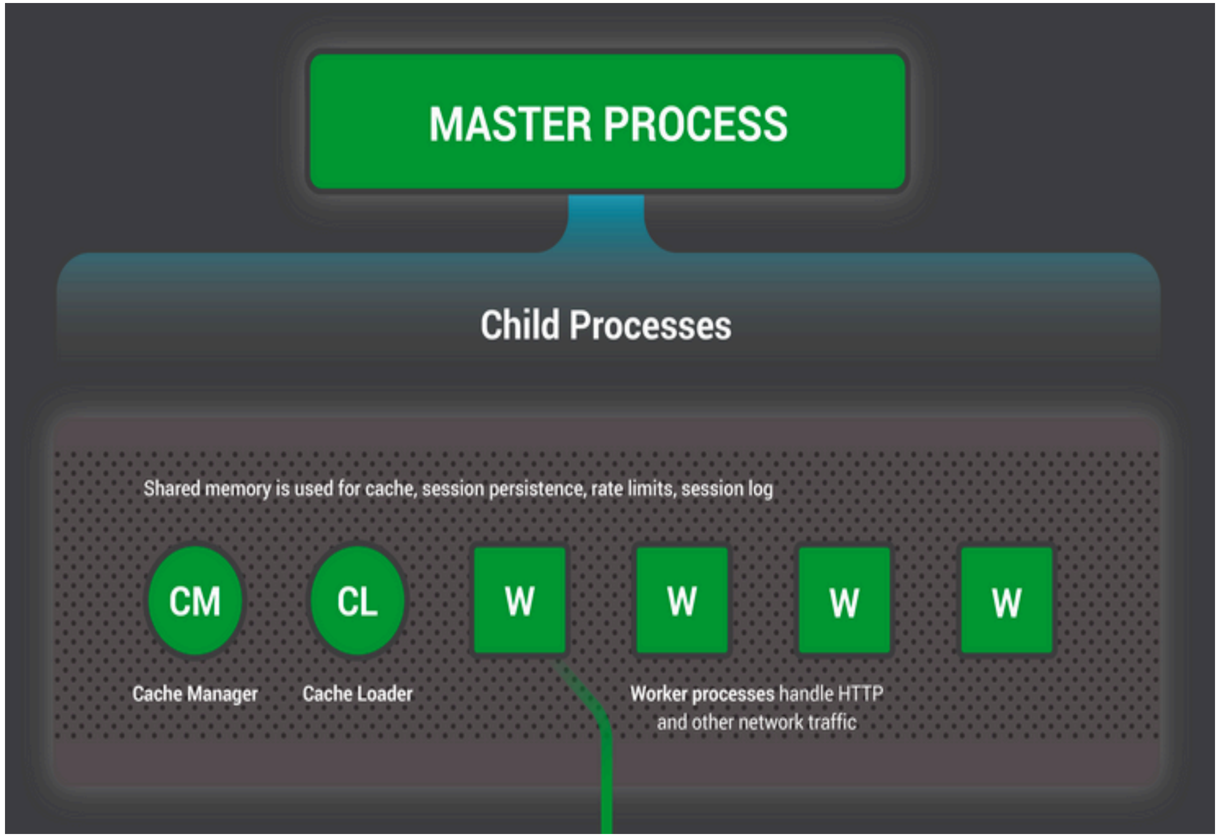
Why NGINX?

- Light Weight : ~10 Mb memory foot print, low CPU Usage
- Concurrent : Supports 100K+ connections
- High performance web server written in C
- Async IO
- Event Driven
- Pluggable architecture
- Native binding to Lua
- Architectural details

Webserver Usage

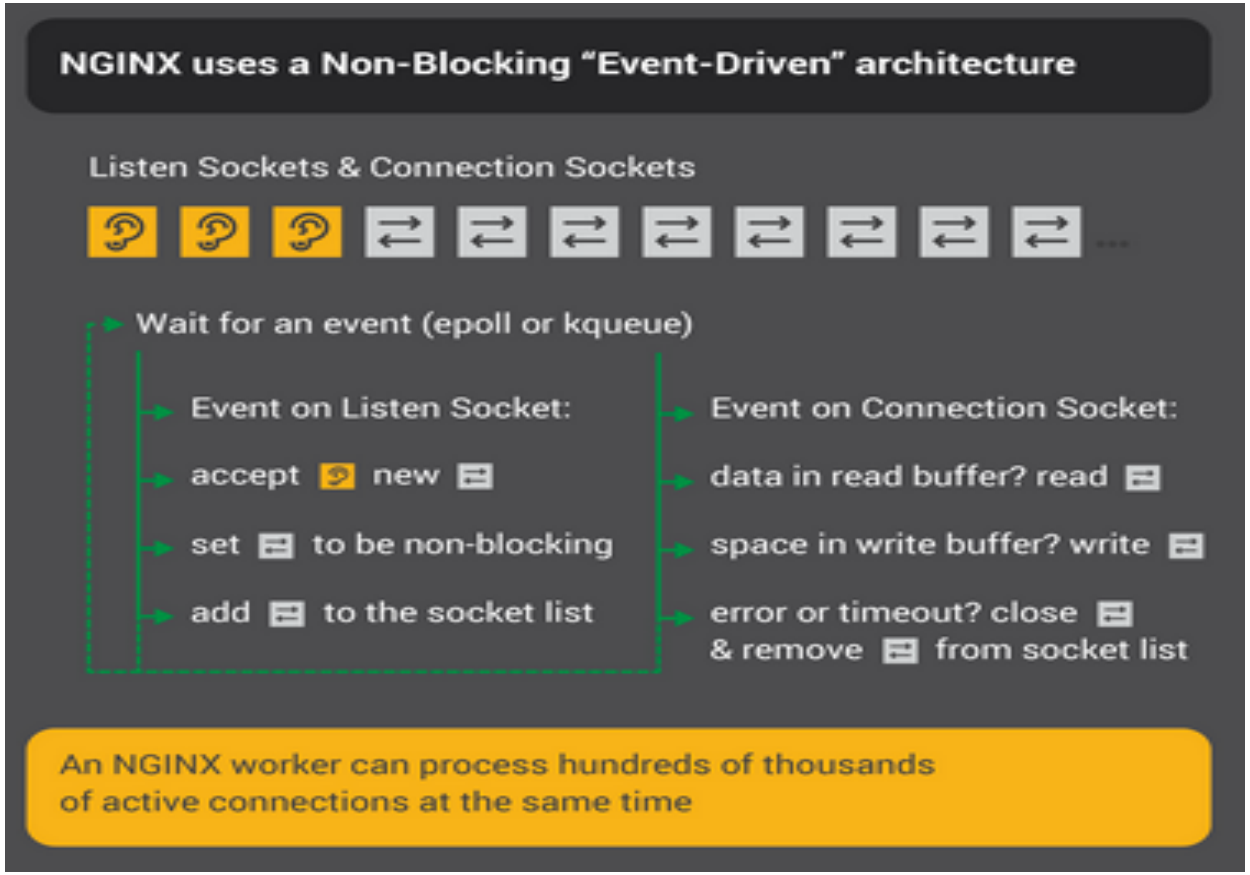


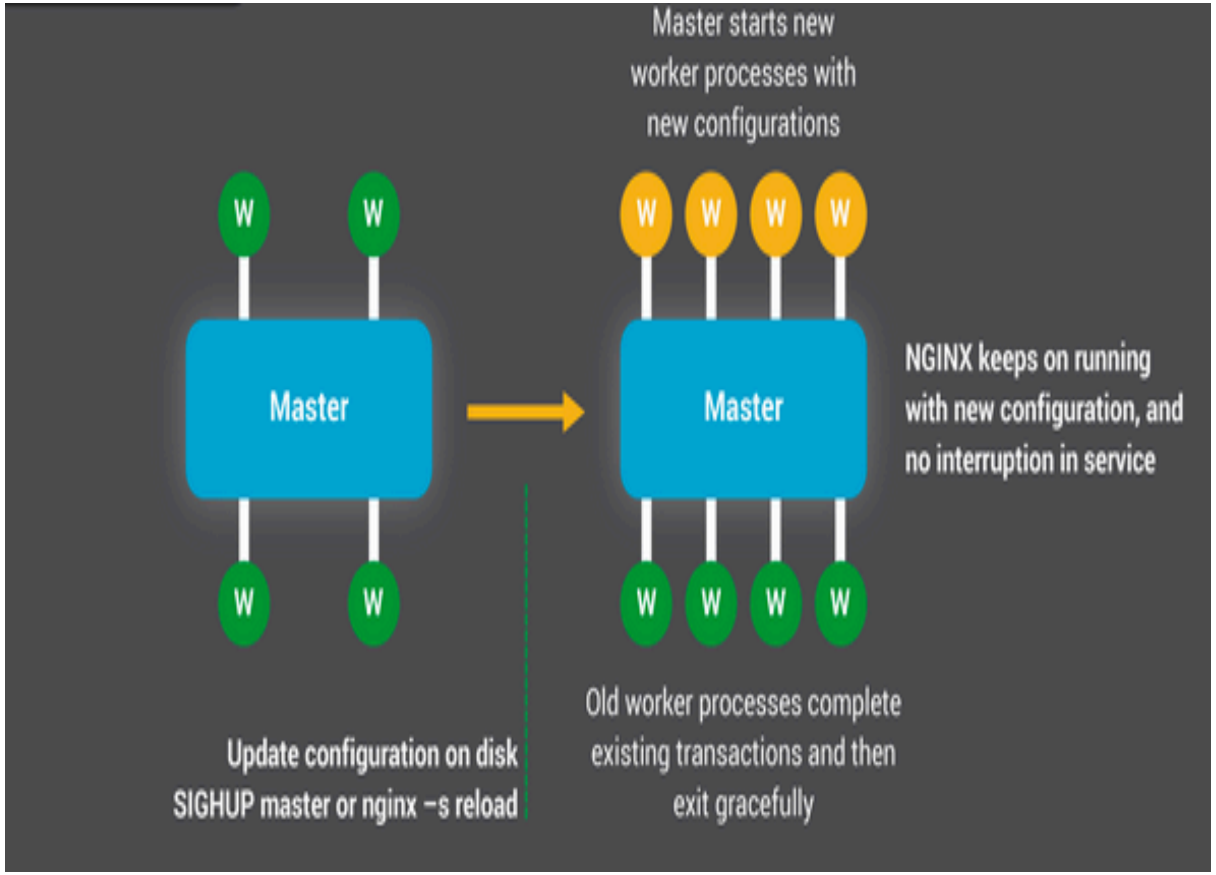
Why NGINX?



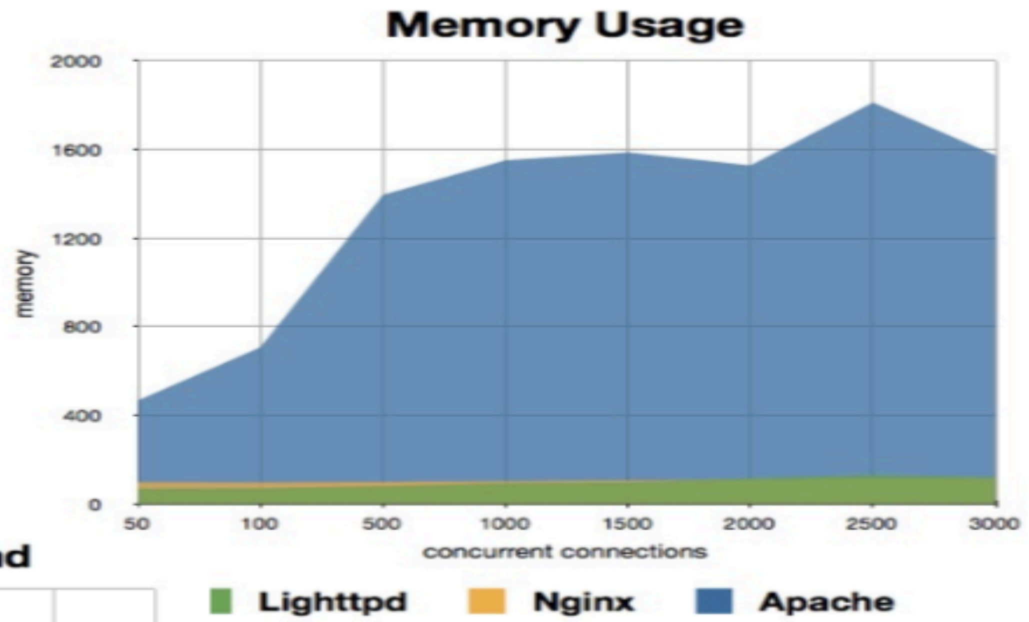
- The *master* process performs the privileged operations such as reading configuration and binding to ports, and then creates a small number of child processes (the next three types).
- The *cache loader* process runs at startup to load the disk-based cache into memory, and then exits.
- The *cache manager* process runs periodically and prunes entries from the disk caches to keep them within the configured sizes.
- The *worker* processes do all of the work! They handle network connections, read and write content to disk, and communicate with upstream servers.

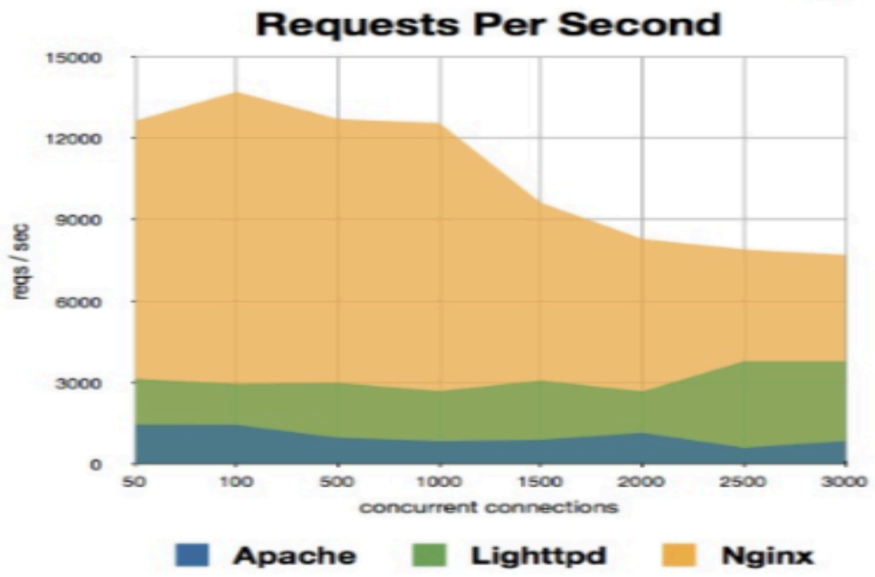
How NGINX works





NGINX is fantastic at Scaling and Handling Concurrent Connections





Lua Language

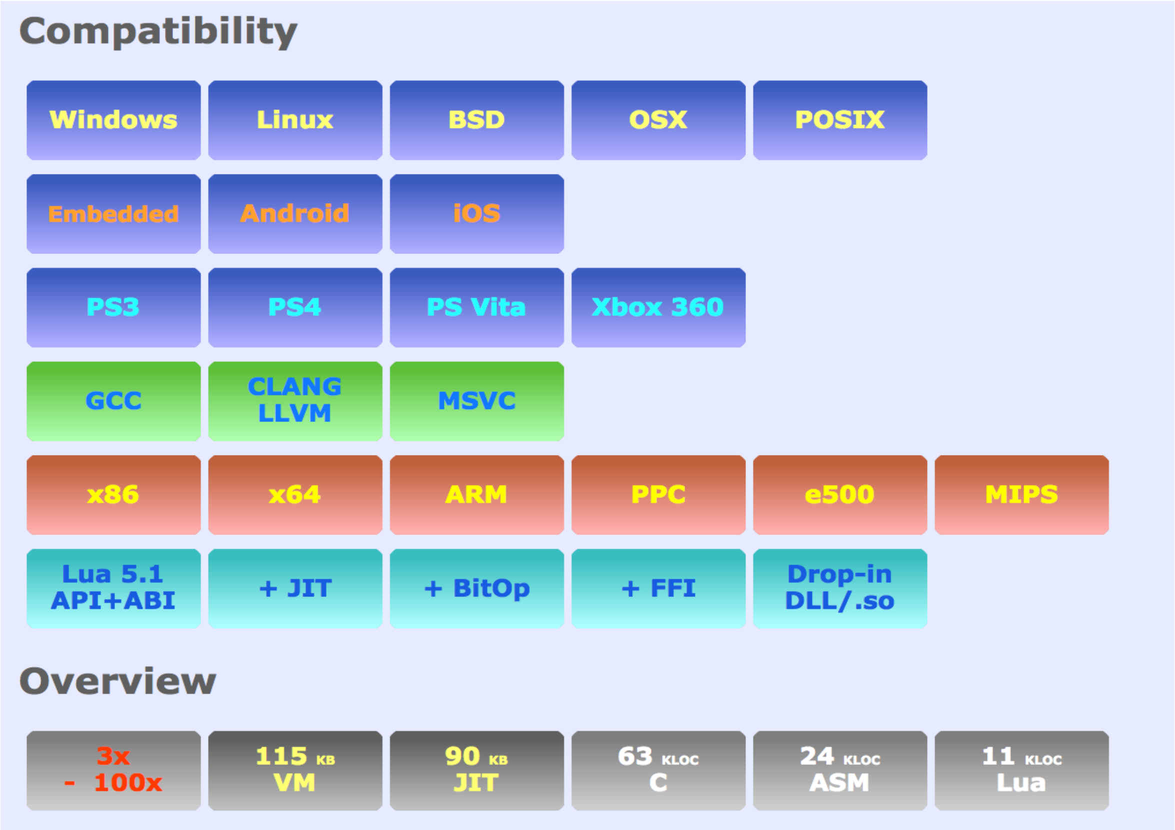
- Lua is a powerful, efficient, lightweight, embeddable scripting language.
- It supports procedural programming, object-oriented programming, functional programming, data-driven programming, and data description Brief description or definition of topic or project
- "Lua" (pronounced LOO-ah) means "Moon" in Portuguese

- Lua is designed, implemented, and maintained by a team at PUC-Rio, the Pontifical Catholic University of Rio de Janeiro in Brazil
- Lua is distributed in a small package and builds out-of-the-box in all platforms that have a standard C compiler and supports embedded to IBM Mainframe

Lua/ NGINX (OpenResty) Offers flexibility of Lua with the power of statically compiled C++

- **LuaJITBinding:ZeroMQBinding (http://zeromq.org/bindings:lua)**
- **LuaJIT2:**
  - mean throughput: 6,160,911 [msg/s]
  - mean throughput: 1478.619 [Mb/s]

- **C++code:**
  - mean throughput: 6,241,452 [msg/s]
  - mean throughput: 1497.948 [Mb/s]



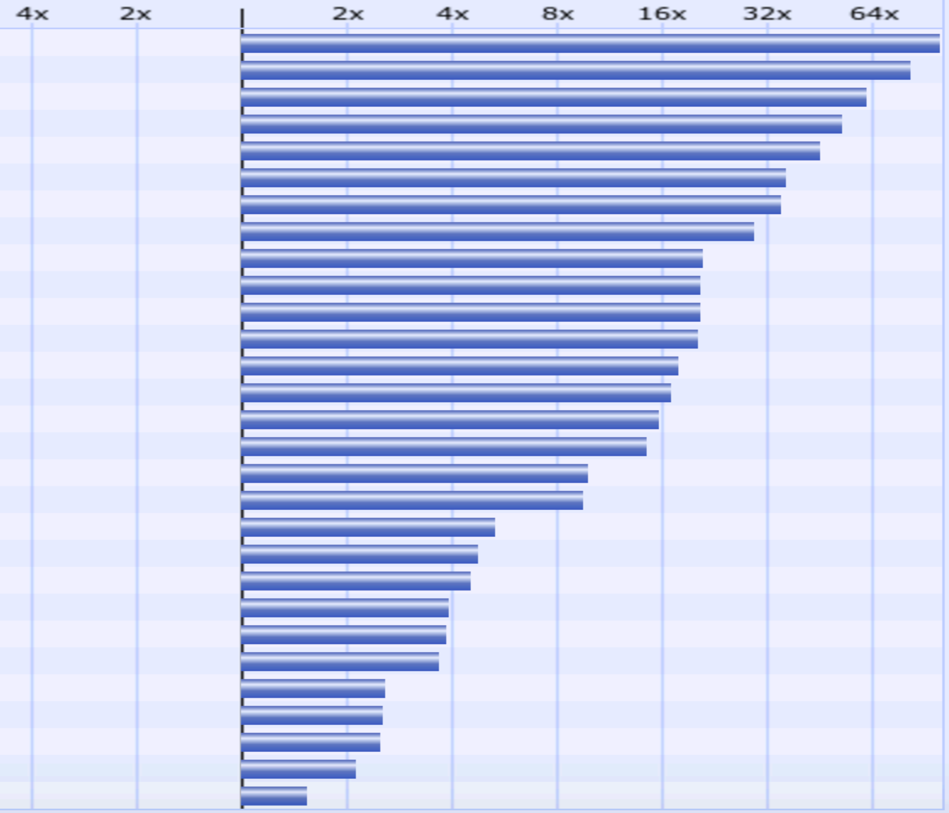
The Lua JIT (Just In Time) Compiler Ensures that our code runs fast

Click to compare	Mode
Lua 5.1.5	x86
LuaJIT 1.1.6 -O	x86
LuaJIT 2.0.0 (interpreter)	x86
LuaJIT 2.0.0	x86
Lua 5.1.5	x64
LuaJIT 2.0.0 (interpreter)	x64
LuaJIT 2.0.0	x64

Interactive Performance Comparison Chart

- Select two VMs to compare. Click to the left for the 1st one and to the right for the 2nd one. Then take a look at the results below.
- E.g. a ratio of 49.71 means the 2nd VM runs that benchmark almost fifty times faster. Please note the bar graph has *logarithmic* scale.
- Choose different VMs to compare or compare the same VM on x86 and x64.
- Click on the arrows next to *Benchmark* or *Ratio* to sort by these columns.

Benchmark▼	N	Ratio▼
md5	20000	112.09
array3d	300	84.00
euler14-bit	2e7	62.02
mandelbrot-bit	5000	53.57
scimark-lu	5000	45.66
scimark-fft	50000	36.84
scimark-sor	50000	35.66
nsieve-bit	12	29.60
spectral-norm	3000	21.09
fannkuch	11	20.94
ray	9	20.89
nbody	5e6	20.43
mandelbrot	5000	18.10
recursive-ack	10	17.24
scimark-sparse	15e4	15.97
recursive-fib	40	14.60
pidigits-nogmp	5000	10.02
nsieve-bit-fp	12	9.56
k-nucleotide	5e6	5.38
binary-trees	16	4.79
fasta	25e6	4.60
nsieve	12	3.97
coroutine-ring	2e7	3.89
partialsums	1e7	3.73
revcomp	5e6	2.61
chameneos	1e7	2.55
life		2.51
series	10000	2.16
sum-file	5000	1.56



Leading Lua Users

- **World of Warcraft** : Multiplayer game
- **Adobe** Lightroom: environment for the art and craft of digital photography
- **LEGO Mindstroms** NXT
- Space Shuttle Hazardous Gas Detection System: *ASRC **Aerospace**, Kennedy Space Center*
- **Barracuda** Embedded Web Server
- **Wireshark** : Network protocol analyzer
- Asterisk: Telecom PBX
- Radvision SIP Stack
- **Redis**, **RocksDB**(Facebook), Tarantool and many other DBs

- Capital One:
  - Capital One DevExchange API Gateway,
  - Virtual Cards
  - Tokenization Platform

Capital One's Restful API & Architecture Journey

Capital One has been investing heavily in RESTful APIs since 2014

- **There was a strong need for a gateway to serve as the single point of entry for all API traffic.**
- **The Gateway handles**
  - Authentication
  - Authorization
  - Rate Limiting
  - API routing
  - Custom Policies

By 2016 we had an opportunity to consolidate a number of legacy gateway products

- **Given our GW consolidation & migration strategy, our requirements grew complex**

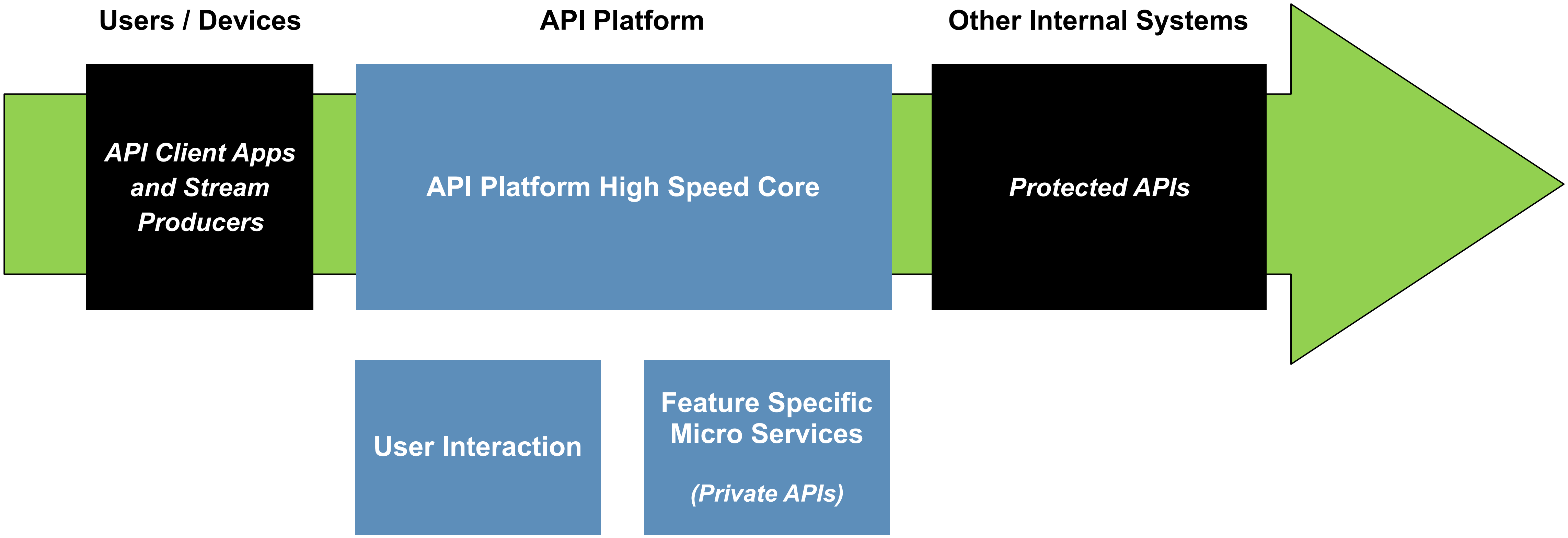
- **12 external options were evaluated**
  - 8 options were eliminated based prior to load testing
  - 4 commercial & open source gateways were load tested head to head
- **In addition, we evaluated Rohit's Prototype**
- **We selected our home-grown solution based on features, performance, resiliency and scalability**

At first I didn't believe Rohit, so I did my own testing

Experimental API to test out the relative performance

Language (Framework)	Multitasking Model Used	Average Throughput
NodeJS (ExpressJS)	Single Threaded Event Loop	~12K TPS
Java (Spring Boot)	Multi-threaded	~15K TPS
Go (Standard Libraries)	GO Routines	~95K TPS
Lua JIT with NGINX	Single Threaded Event Loop	~97K TPS

We separate features based on the Level of performance required



We defined our Architecture / Design Principles to ensure we can meet our high NFRs

- **Leverage ACID transactions only where required and avoid them where possible**
- **Make systems stateless or leverage Immutable data that is safe to cache indefinitely**
- **Separate reads from writes**
- **Partition or Shard Data to meet SLAs**
- **Micro-batch processing**

We Leverage ACID* Transactions Only Where Required and Avoid Them Where Possible

- **Ensuring data consistency is hard:**
- **Data replication and coordination takes time**
- **Examples Requiring ACID Properties:**
  - Issuing Virtual Credit Card Numbers, Issuing New Tokens & Coordinating API changes
- **Examples that don't require ACID Properties:**
  - Logging, Reading of Immutable Data/Tokens

We Make Systems Stateless or Leverage Immutable Data that is Safe to Cache Indefinitely

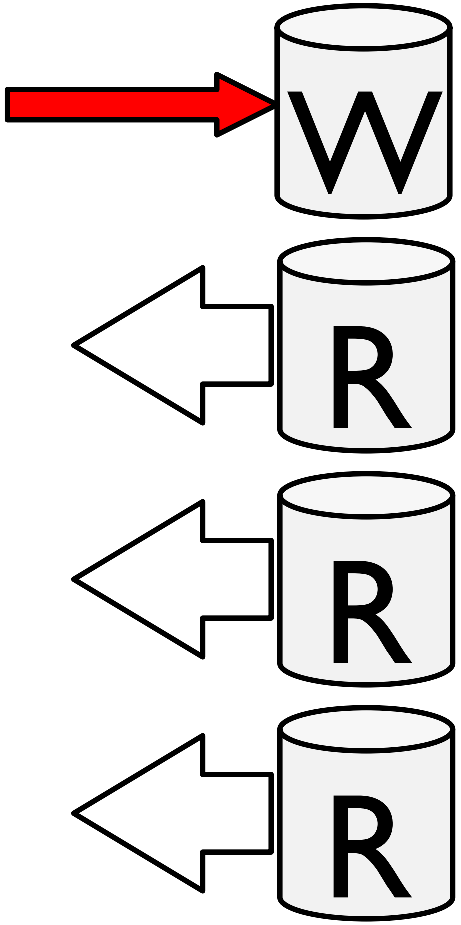
- **Many API Gateways store a copy of Access Tokens in a database**
- **The Token Lifecycle can be broken into 2 pieces to make it scale better:**
  - DevExchange Gateway Issues Stateless JWE Access Tokens
  - Revoking an Access Token can still be Accomplished with a token blacklist
- **For the Tokenization Use Cases are immutable and can be cached permanently on each server**

We Separate Reads from Writes to Scale

- **Separating Reads and Writes can Allow them to be scaled differently without inhibiting the other operation**

- **For the Tokenization use case:**
  – The relationship between Tokens and Original Values are cached on every machine
  – Creating new tokens requires ACID transactions and uses RDS underneath with out of region encrypted read replicas

Workload: 98% Reads / 2%Writes



Partition or Shard Data to meet SLAs

- **Partitioning or sharding data can:**
  - Spread the load
  - Guarantee cache availability
  - Ensure consistent performance

- **Partitioning can be managed manually or provided by the Storage Platform**

- **Tokenization Use Case:**
  - Data is partitioned based on field type (Separate Caches and RDBMs)

Performance Testing

On the same test bed different load testing tools gave very different results:

- SOAP UI: 816 TPS
- AB: 7,088 TPS
- JMeter: 24,332 TPS
- WRK: 50,646 TPS

Capital One

Results

We were able to deliver a full featured API Gateway product to our stakeholders

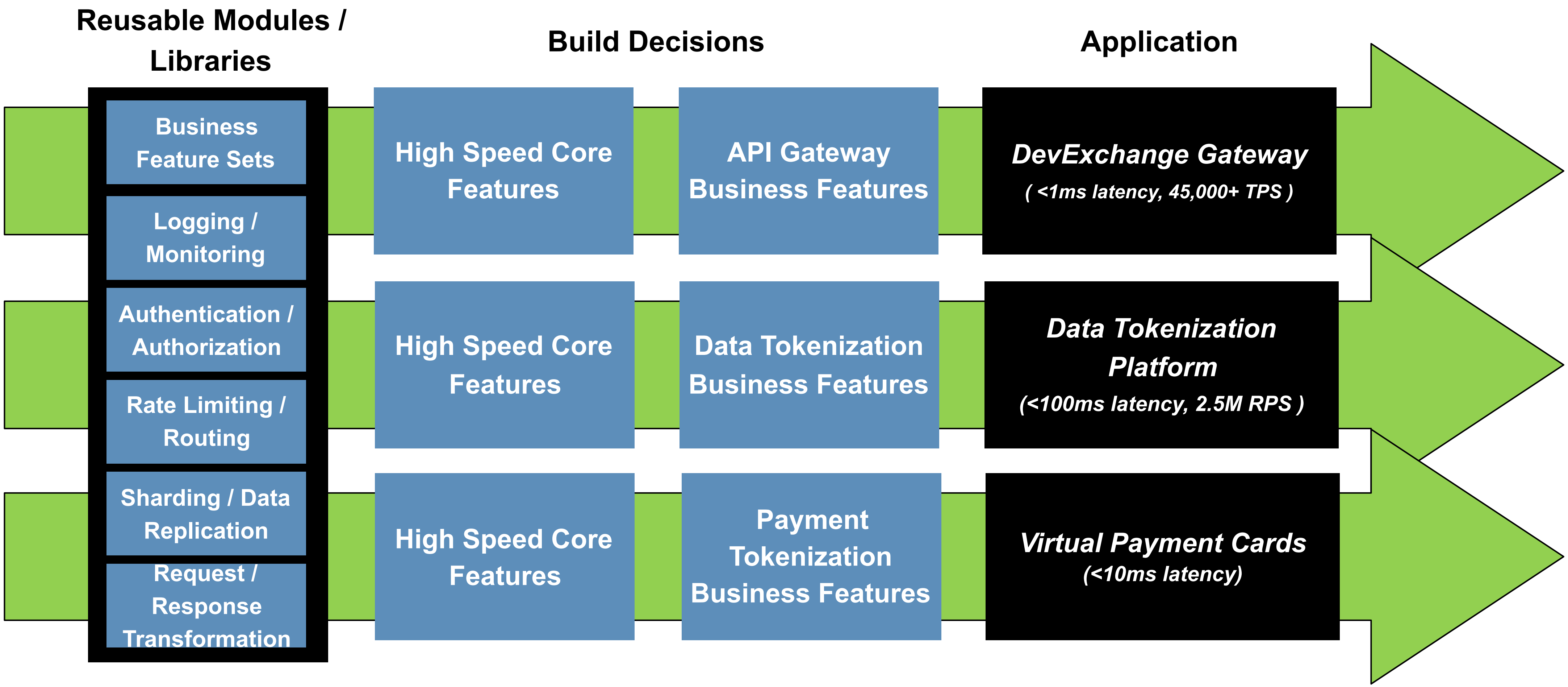
Load Balancing	Routing REST/SOAP	Throttling	HTTP 1.0/2.0 SSL / TLS1.1/1.2
Authentication	OAuth 2.0 / JWE / JWT	Token Inactivity	Token Revoke List
Request Response Transformation	IP White / Black List	Scriptable Policies	HSM Integration
Cloud Automation	Logging	Monitoring	Caching

Tech Stack

LuaJIT	Nginx	Redis	AWS-RDS

CapitalOne

Based on the success of the API Gateway, we built 2 additional systems on the same stack



Our NGINX stack has enabled us to meet and exceed all of our expectations

- **DevExchange Gateway**
  - >2 billion transactions per day
  - 45,000 transactions per second (peak)
  - < 1ms latency (Average)

- **Data Tokenization Platform**
  - 4+ Billion records
  - 3+Terabyte of data
  - 12 billion operations per day
  - 2.5 million operations per second (peak)
  - 20 – 40ms latency (Average)

- **Virtual Payment Cards**
  - < 2 ms latency (Average)

Thank You